Listing 7.33: The `flash` ERb in the site layout using `content_tag`.
app/views/layouts/application.html.erb

```
<!DOCTYPE html>
<html>
      .
      .
      .
      <% flash.each do |message_type, message| %>
        <%= content_tag(:div, message, class: "alert alert-#{message_type}") %>
      <% end %>
      .
      .
      .
</html>
```

7.5 Professional-grade deployment

Now that we have a working signup page, it's time to deploy our application and get it working in production. Although we started deploying our application in Chapter 3, this is the first time it will actually *do* something, so we'll take this opportunity to make the deployment professional-grade. In particular, we'll add an important feature to the production application to make signup secure, we'll replace the default webserver with one suitable for real-world use, and we'll add some configuration for our production database.

As preparation for the deployment, you should merge your changes into the `master` branch at this point:

```
$ git add -A
$ git commit -m "Finish user signup"
$ git checkout master
$ git merge sign-up
```

7.5.1 SSL in production

When submitting the signup form developed in this chapter, the name, email address, and password get sent over the network, and hence are vulnerable to

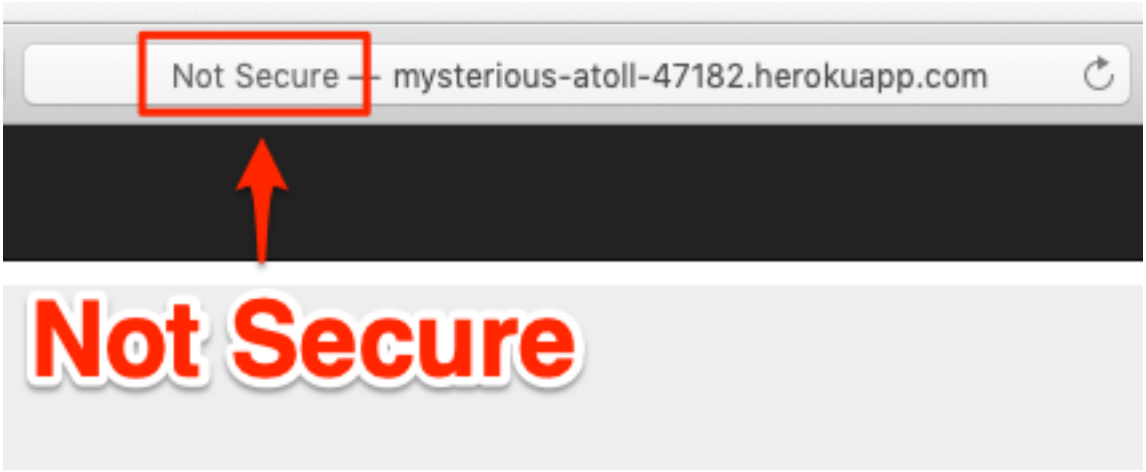

Figure 7.26: The result of using an insecure http URL in production.

being intercepted by malicious users. This is a potentially serious security flaw in our application, and the way to fix it is to use Secure Sockets Layer (SSL)[13] to encrypt all relevant information before it leaves the local browser. Although we could use SSL on just the signup page, it's actually easier to implement it site-wide, which has the additional benefits of securing user login (Chapter 8) and making our application immune to the critical *session hijacking* vulnerability discussed in Section 9.1.

Although Heroku uses SSL by default, it doesn't *force* browsers to use it, so any users hitting our application using regular http will be interacting insecurely with the site. You can see how this works by editing the URL in the address bar to change "https" to "http"; the result appears in Figure 7.26.

Luckily, forcing browsers to use SSL is as easy as uncommenting a single line in **`production.rb`**, the configuration file for production applications. As shown in Listing 7.34, all we need to do is set **`config.force_ssl`** to **`true`**.

Listing 7.34: Configuring the application to use SSL in production.
`config/environments/production.rb`

```
Rails.application.configure do
  .
```

[13] Technically, SSL is now TLS, for Transport Layer Security, but everyone I know still says "SSL".

```
  .
  .
  # Force all access to the app over SSL, use Strict-Transport-Security,
  # and use secure cookies.
  config.force_ssl = true
  .
  .
  .
end
```

At this stage, we need to set up SSL on the remote server. Setting up a production site to use SSL involves purchasing and configuring an *SSL certificate* for your domain. That's a lot of work, though, and luckily we won't need it here: for an application running on a Heroku domain (such as the sample application), we can piggyback on Heroku's SSL certificate. As a result, when we deploy the application in Section 7.5.2, SSL will automatically be enabled. (If you want to run SSL on a custom domain, such as www.example.com, refer to Heroku's documentation on SSL.)

7.5.2 Production webserver

Having added SSL, we now need to configure our application to use a webserver suitable for production applications. By default, Heroku uses a pure-Ruby webserver called WEBrick, which is easy to set up and run but isn't good at handling significant traffic. As a result, WEBrick isn't suitable for production use, so we'll replace WEBrick with Puma, an HTTP server that is capable of handling a large number of incoming requests.

To add the new webserver, we simply follow the Heroku Puma documentation. The first step is to include the `puma` gem in our **`Gemfile`**, but as of Rails 5 Puma is included by default (Listing 3.2). This means we can skip right to the second step, which is to replace the default contents of the file **`config/puma.rb`** with the configuration shown in Listing 7.35. The code in Listing 7.35 comes straight from the Heroku documentation,[14] and there is no need to understand it (Box 1.2).

[14] Listing 7.35 changes the formatting slightly so that the code fits in the standard 80 columns.